AFRL-SR-AR-TR-09-0211

REPORT DOCUMENTATION PAGE

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	6/11/09	Final 3/1/06 - 11/30/08

4. TITLE AND SUBTITLE
Stateful Publish-Subscribe for XML Data Streams

5. FUNDING NUMBERS
FA9550-06-1-0111

6. AUTHOR(S)
Gehrke, Johannes

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Cornell University
Ithaca, NY 14853

8. PERFORMING ORGANIZATION REPORT NUMBER
50064

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR
4015 Wilson Boulevard, Room 713
Arlington, VA 22203-1954

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

With funding from this grant, we obtained deep understanding of stateful publish-subscribe technology with a special emphasis on data in XML format. We made the following four discoveries: (1) Time is crucial when talking about state in a system. We developed a full characterization and axiomatization of the temporal model in event stream processing languages. (2) We developed a novel language with formal semantics for stateful publish-subscribe. (3) We developed novel algorithms for stateful XML publish-subscribe that achieve two orders of magnitude performance improvements over previous work. (4) We developed a novel framework for the joint optimization of the processing of many stateful publish-subscribe queries. Technology developed with this project has been transitioned to Microsoft where a commercial product is being developed.

14. SUBJECT TERMS
Network-oriented computing, publish-subscribe, XML pattern detection

15. NUMBER OF PAGES
10

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Stateful Publish-Subscribe for XML Data Streams

Final Performance Report 2006-2009

PI: Johannes Gehrke
Award No: AFOSR FA9550-06-1-0111
Institution: Cornell University, Ithaca, NY 14853

Objectives

With funding from this project, we addressed the following research problems:

- **Subscriptions beyond publish-subscribe with simple predicates.** In military network-centric applications, simple attribute-value publish-subscribe systems that evaluate subscriptions over individual objects are inadequate. How can we support more expressive subscriptions that accumulate *state* over time?
- **Subscriptions over objects encoded in XML.** The stateful subscriptions that we need to support are not limited to flat data objects with relational schemata; they also need to support objects encoded in XML for data exchange. How can we match subscriptions over objects that are encoded in XML?
- **The tradeoff between expressiveness and performance.** Previous work has shown that we can gain performance at the cost of expressiveness. For example, we know how to scale publish-subscribe systems to a large number of subscriptions and high publication rates. What are other interesting expressiveness/performance tradeoffs? Can we quantify how much performance we lose for given gains in expressiveness?
- **A subscription language.** Since stateful subscriptions are more expressive than stateless subscriptions, we can express a much larger class of interesting subscription queries through them. However, with this increase in expressiveness comes an increase in the complexity of the query language that clients will use to express subscriptions. What is a suitable language for expressing subscriptions?

Executive Summary

With funding from this grant, we worked on obtaining a deep understanding of stateful publish-subscribe technology with a special emphasis on data in XML format. We made the following four contributions:

1. Time is crucial when talking about state in a system. We developed a full characterization and axiomatization of the temporal model in event stream processing languages.
2. We developed a novel language with formal semantics for stateful publish-subscribe.
3. We developed novel algorithms for stateful XML publish-subscribe that achieve two orders of magnitude performance improvements over previous work.
4. We developed a novel framework for the joint optimization of the processing of many stateful publish-subscribe queries.

Technology developed with this project has been transitioned to Microsoft where a commercial product is being developed. We now describe our findings for each of these contributions in detail.

Publications

- Mingsheng Hong, Alan J. Demers, Johannes Gehrke, Christoph Koch, Mirek Riedewald, Walker M. White: Massively multi-query join proeessing in publish/subscribe systems. SIGMOD Conference 2007: 761-772
- Lars Brenna, Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, Walker M. White: Cayuga: a high-performance evcnt processing engine. SIGMOD Conference 2007: 1100-1102
- Walker M. White, Mirek Riedewald, Johannes Gehrke, Alan J. Demers: What is "ncxt" in event proeessing? PODS 2007: 263-272
- Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, Walker M. White: Cayuga: A General Purpose Event Monitoring System. CIDR 2007: 412-422
- Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, Erich J. Neuhold: Proeeedings of the 33rd International Conferenee on Very Largc Data Bases, University of Vienna, Austria, September 23-27, 2007 ACM 2007
- Samuel Madden, Johannes Gehrke: Declarativc, Domain-Speeifie Languages - Elegant Simplicity or a Hammer in Seareh of a Nail? ICDE 2008: 7
- David J. Martin, Johannes Gehrke, Joseph Y. Halpern: Toward Expressive and Sealablc Sponsored Search Auctions. ICDE 2008: 237-246
- Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Henry F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, Gerhard Weikum: The Claremont report on database research. SIGMOD Record 37(3): 9-19 (2008)
- Namit Jain, Shailendra Mishra, Anand Srinivasan, Johannes Gehrke, Jennifer Widom, Hari Balakrishnan, Ugur Çetintemel, Miteh Cherniack, Richard Tibbetts, Stanley B. Zdonik: Towards a streaming SQL standard. PVLDB 1(2): 1379-1390 (2008)
- Minos N. Garofalakis, Johannes Gehrke, Divesh Srivastava: Speeial issue: best papers of VLDB 2007. VLDB J. 18(2): 383-384 (2009)
- Alin Dobra, Minos N. Garofalakis, Johannes Gehrke, Rajeev Rastogi: Multi-query optimization for sketeh-based estimation. Inf. Syst. 34(2): 209-230 (2009)
- Mingsheng Hong, Mirek Riedewald, Christoph Koch, Johannes Gehrke, Alan J. Demers: Rule-based multi-query optimization. EDBT 2009: 120-131

An online demo video shows some of the capabilities of our system; the video is available at:

http://www.cs.cornell.edu/bigreddata/cayuga/cayuga_code/cayuga_demo/cayuga_demo.html

Publications related to this projeet have already been cited over 100 times in the literature according to Google Scholar.

Personnel

The following people worked on research associated with this award:

- Four PhD students worked on systems aspects and research related to this award (Lyublena Antova, Chavdar Botev, Mingsheng Hong, Prakash Linga). Mingsheng Hong's dissertation contains the core research results funded by this grant.
- Lars Brenna was a visiting PhD student from Norway; he worked on several aspects of the Cayuga demo and on parallelization of the system.
- Mohit Tatte was a Master of Engineering student in the Department of Computer Science; he worked on the Cayuga System.
- Dr. Mirek Riedewald, a research associate in group of Gehrke, the PI. In January 2009, Riedewald started as associate professor at Northeastern University; his interview talk was based on research funded by this grant.
- Dr. Walker White. White is a research associate in Gehrke's group; he is an expert on the theory of expressiveness and the design of languages.
- Dr. Alan Demers is a university scientist in Gehrke's group; he is an expert in the design and implementation of scalable systems.
- Professor Christoph Koch is on the faculty in the Department of Computer Science at Cornell University; he is an expert on XML.
- Professor Johannes Gehrke was the PI of the project; he is an expert on the foundations of scalable systems.

Interactions with and Transitions to Industry

There was strong interest in industry in supporting this type of stateful subscription queries for *complex event processing* systems. Mingsheng Hong, the graduate student funded by this award, was twice invited as an intern to Microsoft Research to work on Microsoft's complex event processing research project; this project is now being commercialized within Microsoft. Furthermore, Dr. Mirek Riedewald, one of the leaders of the Cayuga project, has also been invited twice for extended research visits to Microsoft to contribute his expertise to their complex event processing project.

Gehrke led a panel on complex event processing at the 2007 ACM SIGMOD conference. Northrop Grumman and Xerox have started a dialogue with us for technology transfer of some of the Cayuga technology.

The PI (Gehrke) was consultant to Streambase, Inc., the world's leading complex event processing company. Gehrke has advised the company in the area of data stream languages based on work funded by this grant.

Gehrke was also appointed Chief Scientist at Fast Search and Transfer, the world's leading Enterprise Search and Business Intelligence company.

Talks about Work Funded by This Award

- Towards Expressive Publish/Subscribe. Google Inc. New York, NY, June 2006.
- Processing Sensor Streams. Short course at the *Department of Computer Science, University of Aalborg.* Aalborg, Denmark, October 2006.
- Processing Event Streams. Yahoo! Inc. Santa Clara, CA, November 2006.
- Complex Event Processing With Cayuga. Colloquium at the University of Waterloo. Waterloo, Canada, January 2007.
- Processing Data Streams. Invited tutorial at the European Workshop on Data Stream Analysis. Caserta, Italy, March 2007.
- Panel chair for "Data Streams Go Mainstream." Panel at the *2007 ACM SIGMOD International Conference on Management of Data (SIGMOD 2007).* Beijing, China, June 2007.
- Key Trends in Internet Technology. Executive Education, Johnson School of Business, Cornell University. August 2007.
- What's Next in Database Research? Colloquium at the University of Tromsø. Tromsø, Norway, December 2007.
- Panel organizer for "Domain-Specific Languages - Elegant Simplicity or a Hammer in Search of a Nail?" Panel at the 24th International Conference on Data Engineering (ICDE 2008), Cancún, México, April 2008.
- Domain-Specific Declarative Languages. Claremont Retreat of the Database Community. Berkeley, CA, May 2008.
- Scalable Data Stream Processing. Colloquium at the Humboldt-University at Berlin. Berlin, Germany, June 2008.

Major Discoveries

1. A full characterization and axiomatization of the temporal model in event stream processing languages
2. Novel algorithms for stateful XML publish-subscribe that achieve two orders of magnitude performance improvements over previous work
3. A novel framework for multi-query optimization that for the first time enables a convergence of data stream and complex event stream processing

Honors/Awards

The PI, Johannes Gehrke, received a New York State Foundation for Science, Technology, and Innovation Faculty Development Award in 2007.

Major Technical Contributions

1. A Characterization of the Expressive Power of the Temporal Model in Event Stream Languages

Query semanties and implementation efficiency arc inherently determined by the underlying temporal modcl: how events arc sequenced (what is the "next" event), and how the time stamp of an event is represented. Many competing temporal models for event systems have been proposed, with no eonsensus on whieh approach is best. We took a foundational approach to this problem. We created a formal framework presenting event system design choiees as axioms. Thc axioms are grouped into standard axioms and desirable axioms. Standard axioms are eommon to the design of all event systems. Desirable axioms are not always satisfied, but are useful for achieving high performanee.

Given these axioms, we proved several important results. First, we showed that there is a unique model up to isomorphism that satisfies the standard axioms and supports assoeiativity, so our axioms are a sound and complete axiomatization of assoeiative time stamps in event systems. This model requires time stamps with unbounded representations. We presented a slightly weakened version of associativity that pcrmits a temporal model with bounded representations. Wc showed that adding the boundedness condition also results in a unique modcl, so again our axiomatization is sound and eomplete. We believe this model is ideally suited to bc thc standard tcmporal model for complex event processing.

2. A Language for Stateful Subscriptions over XML Data

We designed CEL, thc Cayuga Event Language, a language for stateful subscriptions over XML data that is rich enough to express the eomplex subseriptions that military eommand and eontrol systems requirc. Our language nieely balanees expressiveness and ease of implementation, and it has a well-defined semanties to permit qucry rewrite for query optimization.

The Cayuga Event Language is based on the Cayuga algebra designcd for cxprcssing queries over event streams. It is a mapping of the algebra operators into a SQL-like syntax. Eaeh CEL query has the following simple form:

SELECT < attributes >
FROM < stream expression >
PUBLISH < output stream >

The SELECT clause in CEL is similar to the SQL SELECT elause. It speeifies the attribute names in the output stream sehema, as well as aggregate eomputation. Attributes ean be renamed by AS eonstruets in the SELECT elause. The SELECT elause is optional; omitting it is equivalent to speeifying SELECT *. Thc PUBLISH elause names the output stream, so other queries may refer to it as input. When this clause is omitted, the output stream is unnamed.

We refer to the expression in the FROM elause as a stream expression. This expression is thc corc of eaeh query. A stream exprcssion is composed using a unary construct, FILTER, and two binary eonstruets, NEXT and FOLD. Eaeh of these eonstruets produees an output stream from one or two input streams.

The

FILTER{predExpr}

construct selects those events from its input stream that satisfy the predicate defined by predExpr. CEL, like SQL, is compositional, allowing sub-queries in the FROM clause. We can also publish the outputs of the nested sub-query as a separate stream of its own. We need only add an additional PUBLISH clause to the parenthesized sub-query. This enables one query formulation to produce multiple output streams. Binary constructs in the stream expression allow us to correlate events over time. The first binary construct is

NEXT{predExpr}.

When applied to two input streams S1 and S2 as S1 NEXT{predExpr} S2, this construct combines each event e1 from S1 with the next event in S2 that satisfies the predicate defined by predExpr and occurs after the detection time of e1. When predExpr as the parameter of NEXT construct is omitted, it is by default set to TRUE. For example, S NEXT S returns pairs of consecutive events from stream S. A more powerful use of the NEXT construct exploits what we call "parameterization," the ability of the predExpr to refer to attributes from both its input streams.

In our previous discussion, two input streams of the NEXT construct have disjoint schemas. Of course this is not typical – the two input streams of a binary construct frequently contain identically named attributes. This situation happens to the binary join operator in relational algebra or SQL as well, for example in self-joins. When this happens to a binary construct, the reference to an attribute name in the predicate associated with the construct could become ambiguous, since the attribute could be from either one of the two input streams. To address such reference ambiguity in CEL, we introduce special language constructs, referred to as decorators, to identify the streams from which attributes are taken. $1.foo refers to attribute foo in the first input stream of a binary construct, or the single input stream of a unary construct. Similarly, $2.foo refers to attribute foo in the second input stream of a binary construct. The decorator of an attribute can be omitted when it is in the schema of only one input stream. For simplicity of CEL, decorators are allowed only in predicate expressions, and cannot be used in the SELECT clause. It is helpful to view the SELECT clause as receiving one input stream whose schema is produced by the FROM clause.

While NEXT allows us to correlate two events, there are many situations where we need to iterate over an a-priori unknown number of events until a stopping condition is satisfied. This capability is supplied by the FOLD construct. A FOLD construct has the form

FOLD{predExpr1, predExpr2, aggExpr}.

The three parameters respectively denote (1) the condition for choosing input events in the next iteration; this plays the same role as predExpr in NEXT{predExpr} (2) the stopping condition for iteration, and (3) aggregate computation between iteration steps. Intuitively, FOLD is an iterated form of NEXT that looks for patterns comprising two or more events. As with NEXT, we use decorators $1 and $2 respectively to refer to attributes in the first and the second input streams of FOLD. To refer to attributes in the last iteration of FOLD from the second stream, we use decorator $.

To ensure valid iterations in the FOLD construct, we maintain a schema inclusion invariant that the schema of its first input stream be a superset of the schema of its second input stream. Queries that violate this invariant will be rejected as being illegal. Note that when the two input streams of a binary construct have identically named attributes, without proper renaming, the output stream of the binary construct will have duplicate

attribute names, making the data semantics ambiguous. In relational algebra or SQL, this situation is addressed by explicitly renaming the output attribute names to make them distinct. Similarly for each NEXT construct, explicit renaming can be used to avoid name collisions in its output schema. For the FOLD construct, however, collisions cannot be avoided due to the schema inclusion invariant. We use an automatic renaming scheme as follows to make sure the attributes in the output stream schema have distinct names. It applies to both NEXT and FOLD as follows:

For sub-expression R NEXT{predExpr} S, where R and S denote the input streams of the binary construct NEXT, if R and S do not have attribute name collision, the output schema will be a cross product of the two input schemas of R and S, and no renaming is performed. Otherwise, we rename each attribute a in R to a1. For uniformity, even attribute names in R that do not appear in S are renamed this way. However, no renaming is performed on attribute names of S. It is possible that after this renaming operation, there are still duplicate attribute names in the output schema (consider the case when S has an attribute named a1). In this case, the input query will be rejected as illegal. For sub-expression R FOLD{unaryExpr, predExpr, aggExpr} S, for each attribute a that occurs in both R and S, the value of a in R will be stored in attribute a1 in the output schema of the above sub-expression, and the value of attribute a in the latest iteration of S will be stored in attribute a in the output schema. Each attribute b in R but not in S will still be named b in the output schema.

Note that with attribute renaming, we avoided the use of hierarchical decorators such as $1.$1.foo to refer to attribute foo in the first stream S of expression (S NEXT S) NEXT S. Hierarchical decorators are therefore not allowed in CEL. We believe our renaming scheme, when used appropriately, makes it easier to write queries by rendering explicit renaming unnecessary, and thus improves the user-friendliness of CEL.

3. Algorithms for Scalable Multi-Query Join Processing for Stateful Subscriptions over XML Data

XML has become the primary standard for data exchange on the Internet and for enterprise applications. The rapid emergence of Web Services in particular has underlined the need to support efficient XML processing in distributed environments. A crucial component of Web Service based architectures are message brokers. They manage large numbers of subscriptions, or queries that express the interest of subscribers—both users and applications. The subscriptions are matched in real-time with event streams (or for short, streams) of incoming XML documents, created by publishers like applications behind a Web Service interface, news services, or blog writers. Because of its close relationship to traditional publish/subscribe (pub/sub) systems, we will use the term XML publish/subscribe system to refer to this class of message brokers. In the setting of processing XML streams, events and documents are interchangeable terms.

It is crucial for XML pub/sub systems to be both expressive and scalable. Expressiveness refers to the ability of the query language to support a wide variety of queries. The downside of greater expressiveness is that complex queries are more difficult to implement efficiently. For applications like message brokering, an XML pub/sub system has to scale in terms of the number of subscriptions and the stream rate of incoming messages, while providing sufficient functionality to express all relevant subscriptions.

There had been much work on XML pub/sub systems that can efficiently process a large number of XML subscriptions over streaming XML documents. These systems support a proper subset of XPath 1.0, typically limited to forward axes (child and descendant), predicate evaluation and wild card operator *. However, they are unable to express a large class of important queries: queries that correlate multiple input events to detect complex patterns in real-time as required for stateful publish/subscribe. This class has been recognized as being highly important for event processing. We refer to these queries as inter-document queries. Inter-document queries join different XML documents based on values in their nodes, either attributes or text. An inter-document query is capable of joining multiple documents in either the same XML stream, or across multiple

streams. For example, for monitoring blogs and news articles, users might be interested in blog postings by the same author or about the same topic that appear within a short time of each other and are above some reputation threshold. Inter-document queries are also building-blocks for more powerful queries like finding all electronics product announcements that "create above-average attention in the blogosphere." In enterprises, related events containing information about the quality of service that customers receive need to be processed to monitor compliance with service level agreements.

XML message brokers are used for applications ranging from tens of publishers and subscribers, in small enterprises, to hundreds of thousands of users in Internet scale RSS feed monitoring for blogs and news. Hence an XML pub/sub system has to process anywhere from a few hundred to millions of concurrently active subscriptions for streams that can have high arrival rates. The only way to achieve this kind of scalability is by effective multi-query optimization (MQO). Unfortunately, MQO for inter-document queries is a very challenging problem. In an inter-document query, the join condition consists both of tree and node value comparisons. This can create a wide variety of conditions with little apparent commonality.

To address this issue, we dissect each query into tree pattern components and value comparison components. The tree pattern components are expressible in the simpler XPath fragments supported by existing XML pub/sub systems like YFilter. This enables us to leverage existing XML pub/sub technology for efficient discovery of tree pattern components. Unfortunately this does not suffice, because the main performance bottleneck in practice is the evaluation of the value comparison components, as is confirmed by our experimental results.

Our research shows that value comparison components, which have only very limited structure information, almost always can be described by a small number of query templates. This is guaranteed for XML documents that have a fairly regular schema, which is common in practice, and for documents with a small number of nodes, which is often the case for individual RSS feed items. Even for other XML streams, in practice the number of value comparison components is small, because only a few of the possible comparisons are semantically meaningful. This observation gives us a powerful handle on MQO. Without dissecting join conditions, each different condition would have to be implemented and executed individually, similar to a nested loops join whose outer loop iterates over all queries and whose inner loop evaluates the join predicates. Our dissection approach induces a partitioning of the query set into a small number of equivalence classes, one for each query template. Now we only need a per-template implementation and can take advantage of set-oriented processing of all queries that belong to the same template. By mapping this into a relational join problem, we can take advantage of a wealth of expertise in relational query processing.

The query dissection into tree pattern and value comparison components naturally leads to a two-stage approach to query processing. Our system has two major components—the XPath Evaluator for processing all tree pattern components and the Join Processor for evaluating the value comparison components. For an incoming XML document, first the XPath Evaluator is invoked to evaluate the tree patterns. It produces a set of bindings of variables defined in these patterns. These bindings are referred to as XPath witnesses, or witnesses for short. Second, the Join Processor uses the witnesses to perform value joins on a per-template basis. In this scheme, the XPath Evaluator can be viewed as an access method or accelerator for efficiently "retrieving" the witnesses for the join processing stage.

Our experiments have shown that our novel approach achieves *two orders of magnitude performance improvements* over the state of the art.

4. A Methodology for Rule-Based Multi-Query Optimization for Stateful Publish-Subscribe and Data Stream Processing Systems

Query optimizers have been instrumental for the success of relational database technology. The cost difference between a good and a bad query execution plan can be several orders of magnitude. For systems with stateful queries the stakes are even higher. Instead of one-shot queries in a relational DBMS, a stateful publish-subscribe (for short, data stream0 system processes many continuous queries simultaneously. These queries are active for long periods of time and they process massive streams in real time. A poor query implementation choice can negatively affect system performance for the lifetime of the query.

The key to achieving good stream processing performance is to optimize multiple queries together, rather than individually. In a stream query workload, it is often the case that multiple concurrently active queries can share state and computation. Query evaluation techniques that exploit this property are referred to as Multi-Query Optimization (MQO) techniques. The importance of MQO for stream processing is widely accepted and various stream MQO techniques have been proposed in the literature. Unfortunately, existing MQO techniques apply only to very specific queries or workload properties. For example, predicate indexing is tailored for a set of selection operators that all read the same input stream. In addition, work on MQO techniques so far has happened in parallel for CQL-style stream engines, referred to as Relational Engines (RE), and event pattern detection engines, referred to as Event Engines (EE). The former use an operator model similar to relational databases, while the latter (such as our Cayuga System) implement queries with automata. This has led to an unsatisfactory state of MQO, characterized by a confusing variety of individual techniques that apply to specific workloads or implementation models only. This prevents effective MQO for complex queries and leads to a situation where similar approaches might be re-invented by the different communities for REs and EEs.

To address these problems, we developed a Rule-based MQO framework, called RUMOR. It is inspired by the classical Query Graph Model (QGM) of RDBMSes, where query optimization techniques for single queries can be naturally modeled as transformation rules on query plans. RUMOR provides a modular and extensible framework, enabling new optimization techniques to be developed and incorporated incrementally into the system.

To support rule-based MQO, we have to extend the key abstractions that are used in a traditional RDBMS or stream system: physical operators, transformation rules, and streams. We introduce a small number of carefully designed abstractions that together create a powerful MQO framework. In fact, RUMOR incorporates all previously proposed MQO techniques for stream processing. In particular, it successfully incorporates MQO techniques from both relational stream engines and automata-based event processing engines. Hence an additional benefit of RUMOR is that it enables the unification of these diverse camps of stream processing systems. Experimental results for our prototype implementation indicate that we can efficiently process a large number of CQL-style relational stream queries, event processing queries, as well as hybrid queries involving query features from both types of query workloads.

Thus RUMOR lays a new foundation for multi-query optimizers (MQOptimizers) for data stream processing. It has also opened up opportunities for exciting future research on finding new rewrite rules and on extending the approach to cost-based MQOptimizers, incorporating ideas from the classical dynamic programming approach to cost-based single query optimization in RDBMSes